Express Mail Label No. EV041360286US

ENTRYPOINT DISCOVERY TECHNIQUES IN A BIOS ENTITY

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] Not applicable.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[0002] Not applicable.

BACKGROUND OF THE INVENTION

Field of the Invention

[0003] The preferred embodiments of the present invention relate to operation of computer systems. More particularly, the preferred embodiments are directed to calling basic input/output system (BIOS) routines in a computer system. More particularly still, the preferred embodiments are directed to determining the availability and the specific interrupt service number of BIOS routines in a computer system.

Background of the Invention

[0004] The heart of any computer system is the motherboard. The motherboard generally contains a microprocessor, main memory array, and various bridge devices which enable hardware and software components to communicate and perform their respective functions. Given the fact that there are many motherboard manufacturers, which may include any of an array of possible microprocessors, there is likewise an array of varying steps and procedures required for software to communicate with hardware. However, software applications, for example word processors and internet browsers, are typically written for a class of computers without distinction as to a specific brand of computer system or the specific type of motherboard. Thus, some functions which software applications perform are abstracted (or taken away from) the software applications such that the specifics for each hardware implementation need not be embodied in the user-level software. The first level of abstraction between user-level programs and hardware typically takes the form of driver programs. Thus, rather than user-level software attempting to communicate directly with hardware, this software need only be programmed to communicate with a respective driver program. The driver program then is assigned the task of communicating with the hardware devices. Performing the task of communicating with hardware may take the form of calling basic input/output system (BIOS) routines to perform very specific tasks. While many BIOS routines are standard across all types of computing systems, most ROM BIOS manufacturers allow original equipment manufacturers (OEMs), companies who make the computers that consumers purchase, to define their own BIOS routines. The problem, however, is keeping track of the many BIOS routines that an OEM may provide.

[0005] It has become customary in the art for one person within an OEM to take responsibility for keeping track of the BIOS routines added. That is, if a software developer needs to create a new BIOS routine, the developer contacts the person responsible for managing the added BIOS routines, the BIOS call owner, to request the next available services number. This services number is assigned to the exclusion of all others thereafter. In large corporations, merely finding the person who is responsible for managing the assignment of services numbers for new BIOS routines may be difficult. Moreover, once a BIOS routine has been created under a particular services number, it is very difficult to change or update the services number because of support for legacy driver software that may know the BIOS routine by interrupt category and services number only.

Thus, it is possible that multiple assigned interrupt services numbers (a limited resource) may be allocated to multiple BIOS routines whose functions are only minimally different.

[0006] Outside the context of development and more in the context of operating software, it is difficult for device drivers, software typically responsible for executing BIOS routines, to determine whether a particular OEM-added BIOS routine is supported on a computer system. A first way to determine if a particular BIOS routine is supported is to simply call the BIOS routine and check the return status. Generally speaking, a return value of 86h (86 hexadecimal) from a called BIOS routine is an indication that the BIOS routine is not supported. Calling the BIOS routine in an effort to determine whether that routine is available is dangerous in some cases in that the call to the unsupported BIOS routine may cause a computer system crash.

[0007] A second method for determining whether a particular BIOS routine is supported is to have the kernel level software (driver) who does the BIOS call first check a BIOS information table to obtain a BIOS version number, as well as the BIOS date, and compare the information to determine whether the BIOS supports the BIOS routine desired. This, however, cannot guarantee that in every circumstance a correct conclusion regarding support for a particular BIOS routine, and further may require a substantial amount of information to be hard coded in each driver for the comparison step, which significantly increases the footprint of the driver in main memory.

[0008] Thus, what is needed in the art is a mechanism to uniquely identify BIOS routines, as well as a mechanism to determine whether any particular BIOS routine is supported in a computer system, without the possibility of crashing the system in the determination process or requiring driver programs to keep large tables of information for compatibility checking purposes.

BRIEF SUMMARY OF THE SOME OF THE PREFERRED EMBODIMENTS

[0009] The problems noted above are solved in large part by a method of operating a computer system in which basic input/output system (BIOS) routines supplied by an original equipment manufacturer (OEM) are each preferably assigned a unique identification number. Kernel level software, for example drivers, preferably access a data table, stored in the read-only memory (ROM), to determine a service number associated with a particular BIOS routine (which the driver identifies by the unique identification number for the BIOS routine). Thus, by checking the data table, the driver determines not only whether the particular BIOS supports the desired routine, but also the routine's service number.

[0010] More specifically, preferably each BIOS routine supplied by an OEM is assigned a Globally Unique Identifier (GUID). GUIDs are preferably created with an algorithm that uses the network address and current time of the machine executing the program, and creates a 128 bit number that is, for all practical purposes, globally unique across time and space. Each BIOS routine is then associated with a GUID, and preferably only the supported GUIDs are present in the table in the ROM. The calling driver program checks the table, using the GUID of the desired BIOS routine, to ascertain whether the BIOS routine is supported. If the BIOS routine is supported, the data table also reveals the service number associated with the BIOS routine. Once this information is ascertained by the driver program, the driver initiates the interrupt, including the service number, to start the desired BIOS routine.

[0011] The disclosed methods and structures comprise a combination of features and advantages which enable it to overcome the deficiencies of the related art devices and systems. The various characteristics described above, as well as other features, will be readily apparent to those skilled in the art upon reading the following detailed description and by referring to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] For a detailed description of the preferred embodiments of the invention, reference will now be made to the accompanying drawings in which:

[0013] Figure 1 shows a computer system constructed in accordance with the preferred embodiments; and

[0014] Figure 2 shows, in graphical form, the relationship between an operating system device driver, BIOS routines in a computer system, and the data table of the preferred embodiments which correlates GUIDs to service numbers.

NOTATION AND NOMENCLATURE

[0015] Certain terms are used throughout the following description and claims to refer to particular system components. As one skilled in the art will appreciate, computer companies may refer to a component by different names. This document does not intend to distinguish between components that differ in name but not function.

[0016] In the following discussion and in the claims, the terms "including" and "comprising" are used in an open-ended fashion, and thus should be interpreted to mean "including, but not limited to…". Also, the term "couple" or "couples" is intended to mean either an indirect or direct electrical connection. Thus, if a first device couples to a second device, that connection may be through a direct electrical connection, or through an indirect electrical connection via other devices and connections.

[0017] In most computing systems prior to the Intel™ Itanium™ system (that is, 32 bit or less microprocessors), BIOS routines are generally referred to by an interrupt number, for example interrupt 15h (15 hexadecimal), and a service number, for example service 3. In Itanium™ based computer systems, access to BIOS routines is made through a system abstraction layer (SAL) by means of a C programming language style function call, with the service number identifying the specific routine. Throughout this specification and in the claims, the term BIOS routine means the software program which is identified in the BIOS by a combination of an interrupt and services number, for systems prior to the Itanium™ microprocessors, and strictly by a service number for the Itanium™ based systems (although the preferred embodiments also identify a BIOS routine with a unique identification number). The interrupt number for pre-Itanium™ implementations may also be referred to as an interrupt category. Moreover, the service number may alternatively be referred to as a BIOS call service number or a BIOS routine service number.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0018] Figure 1 shows a computer system 100 constructed in accordance with the preferred embodiment. Computer system 100 generally comprises a microprocessor or CPU 20 coupled to a main memory array 26 and various other peripheral computer system components through an integrated host bridge 22. The CPU 20 preferably couples to the host bridge logic 22 by way of a host bus 24, or the host bridge logic 22 may be integrated into the CPU 20. The CPU 20 preferably comprises an Itanium™ microprocessor made by Intel Corporation. It should be understood, however, that computer system 100 could comprise other microprocessors as well, for example Pentium III™ or Pentium IV™ microprocessors made by Intel Corporation, or microprocessors made by other manufacturers such as Advanced Micro Devices. Thus, the computer system may implement other bus configurations or bus bridges in addition to, or in place of, those shown in Figure 1. Moreover, computer system 100 could also comprise several microprocessors, and this too would be within the contemplation of this invention

[0019] Main memory 26 preferably couples to the host bridge 22 through a memory bus 28. The host bridge 22 preferably includes a memory control unit (not shown) that controls transactions to the main memory 26 by asserting the necessary control signals during memory accesses. The main memory 26 functions as the working memory for the CPU 20 and comprises a conventional memory device or array of memory devices in which programs, instructions and data are stored. The main memory 26 may comprise any suitable type of memory such as dynamic random access memory (DRAM) or any of the various types of DRAM devices such as synchronous DRAM (SDRAM), extended data output DRAM (EDO DRAM), or Rambus™ DRAM (RDRAM).

[0020] The computer system 100 also comprises a graphics controller or video driver card 30 that couples to the host bridge 22 via an Advanced Graphics Port (AGP) bus 32, or other suitable type of bus. Alternatively, the video driver card may couple to the primary expansion bus 34 or one of the secondary expansion buses, for example, PCI bus 40. Graphics controller 30 further couples to a display device 32 which may comprise any suitable electronic display device upon which any image or text can be represented.

[0021] The computer system 100 also comprises a second bridge logic device 36 that bridges the primary expansion bus 34 to various secondary buses including a low pin count (LPC) bus 38 and a peripheral component interconnect (PCI) bus 40. In accordance with the preferred embodiment, the bridge device 36 comprises an Input/Output Controller Hub (ICH) manufactured by Intel Corporation. Although the ICH 36 is shown in Figure 1 only to support the LPC bus 38 and PCI bus 40, various other secondary buses may be supported by the ICH 36.

[0022] In the embodiment shown in Figure 1, the primary expansion bus 34 comprises a Hub-link bus which is a proprietary bus of Intel Corporation. However, computer system 100 is not limited to any particular type of chipset and thus the primary expansion bus may comprise any other suitable buses.

[0023] Referring still to Figure 1, a firmware hub 42 couples to the ICH 36 by way of the LPC bus 38. The firmware hub 46 preferably comprises read only memory (ROM) which contains software programs executable by the CPU 20. The software programs comprises programs to implement basic input/output system (BIOS) commands and instructions executed during and just after power on self test (POST) procedures. The software programs perform various functions including verifying proper operation of various system components before control of the system is turned over to the operating system.

[0024] The preferred embodiments of the present invention are implemented in an Itanium™ based computer system. Itanium™ based systems have migrated away from traditional software interrupts to invoke BIOS routines, and instead use C programming language style calls, with a particular BIOS routine identified by a passed parameter. In particular, consider the following exemplary calling convention as specified by Intel Corporation:

```
typdef struct tagRetParam
{
        INT64        p0;
        INT64        p1;
        INT64        p2;
        INT64        p3;
} rArg;

rArg SAL_PROC (
        WIDTH64BIT        Arg0,
        WIDTH64BIT        Arg1,
        WIDTH64BIT        Arg2,
        WIDTH64BIT        Arg3,
        WIDTH64BIT        Arg4,
        WIDTH64BIT        Arg5,
        WIDTH64BIT        Arg6,
        WIDTH64BIT        Arg7
        ){}
```

Thus, in calling the SAL_PROC procedure eight parameters are passed, with Arg0 being the service number of the desired procedure. Data is returned from the SAL_PROC procedure in the rArg structure.

[0025] At the assembly language level for microprocessors prior to the Itanium™, calling a BIOS routine involves loading an appropriate BIOS call service number into a designated register, and then issuing a software interrupt. The following is an exemplary assembly language syntax program to invoke interrupt 15h (15 hexadecimal), service 1:

```
MOV  AH,1          ; Specify service 1
INT  15h           ; invoke BIOS routine category 15h
```

As the comments to the side of the exemplary assembly language code indicate, the first step may comprise moving a service number into the AH register. Although only the number 1 is used for the service number in this example, one of ordinary skill in the art understands that there may be hundreds or thousands of services associated with a BIOS category (interrupt 15h in the example). After loading the particular service desired in the AH register, the services are invoked which causes the BIOS routines, themselves simply software, to execute on the microprocessor and perform the desired task. To the extent the BIOS routine returns any information, this may be placed in the microprocessor's register or registers as required. The preferred embodiments address determining the proper service number for a desired BIOS routine.

[0026] BIOS routines are short programs to perform very specific tasks, generally associated with the movement of data into or out of the computer system. As discussed in the Background section, performing tasks with BIOS routines abstracts platform independent software routines from the specific hardware implementations. The chipset and/or BIOS manufacturer may specify and provide many BIOS routines, and thus their service numbers are constant across the industry.

However, chipset manufacturers and/or BIOS development companies also allow original equipment manufacturers (OEMs) to specify and provide their own BIOS routines for OEM specific value added tasks.

[0027] In the preferred embodiment, each BIOS routine is assigned a unique identification number, different than the routine's interrupt category and services number, which calling programs, preferably kernel mode drivers, use to identify the desired BIOS routine. Preferably, the unique identifier for each BIOS routine is of a sufficient number of bits that, for practical purposes, no duplicate identification numbers should exist, even between different OEMs. Stated otherwise, the identification numbers are preferably globally unique. In the preferred embodiments, the identification numbers are Globally Unique Identifiers (GUIDs) generated using the program GUIDGEN.EXE, provided with Microsoft developers tool kit. The GUIDs generated with the GUIDGEN.EXE program are 128 bit identification numbers which are based on the network address of the executing computer, the date and time of running the GUIDGEN.EXE program, along with a random number from a random number generator. One of ordinary skill in the art is aware of the GUIGEN.EXE program for use in assigning unique identifiers for software programs outside the context of ROM BIOS routines. Thus, one of ordinary skill in the art, now understanding the preferred embodiment of using the GUIDGEN.EXE program in the context of the BIOS routines, could easily understand their application in the BIOS routine context.

[0028] Thus, each OEM provided BIOS routine is preferably assigned a GUID. However, as exemplified above, BIOS routines are identified by an interrupt category and a service, not the GUID, and thus there should be some mechanism to equate or correlate a GUID to a particular service in an interrupt category. In the preferred embodiments, equating a GUID to a BIOS routine service number is accomplished by way of a look-up table.

[0029] Figure 2 shows, in an exemplary graphical form, the relationship between a calling kernel mode software program, the preferred table for association of GUIDs to service numbers, and execution of BIOS routines. The horizontal line 54 graphically represents the separation of the operating system (OS) 56 and driver 46 from the routines and table 44 preferably stored in the BIOS ROM 42. In the preferred embodiments, kernel mode drivers 46 preferably first access the data or look-up table 44 to determine the service number of a desired BIOS routine (this step indicated by a number 1 in Figure 2). Preferably, the driver searches the table based on the GUID of the BIOS routine desired. For purposes of illustration, consider that the GUID of interest is the GUID held in block 48 of table 44, labeled GUID 2. It must be understood, however, that table 44 does not necessarily contain the actual characters "GUID 2," but preferably contains the 128 bit unique identification program created by the GUIDGEN.EXE program. The GUID 2 location in the table corresponds or correlates to a service number 2. Thus, the driver 46 preferably determines, based on the GUID of the BIOS routine desired, the service number. Once the driver 46 determines the proper service number for the desired BIOS routine, the driver executes the software interrupt in some fashion, for example by calling the desired procedure with the SAL_PROC procedure, passing the proper service number in Arg0.

[0030] A portion of the BIOS code known as a dispatcher 50 preferably analyses the service number passed in the SAL_PROC parameters (or if implemented in an pre-Itanium™ system, the service number stored in the AH register after issuance of the interrupt) and executes the BIOS routine identified by the service number. In the exemplary case of service number 2, the software that performs the desired functions, exemplified by box 52, is called by the dispatcher. After execution of the desired BIOS routine, the dispatcher returns, with or without a return parameter based on the specific type of BIOS routine called.

[0031] Although there may be many ways for a driver program to ascertain the precise location of the table which correlates the GUIDs to the service numbers, in the preferred embodiments the driver programs access the industry standard System Management BIOS (SMBIOS) table to ascertain the location of the data table. One of ordinary skill in the art is well aware of how to locate and access information in the SMBIOS table. Preferably, the SMBIOS table has an entry which gives a pointer to the location of the data table that correlates the GUIDs to the service numbers. While it is preferred that the driver programs access the SMBIOS table to make a determination as to the location of the GUID to service number correlation table, it is also within the contemplation of this invention that the driver programs may also know the location of the correlation table directly.

[0032] Referring again to Figure 1, the firmware hub 42 is shown to have a table 44 to exemplify that preferably the table used to correlate the GUIDs to service numbers in any particular machine is preferably stored on the firmware hub. As one of ordinary skill in the art is aware, BIOS routines stored on the firmware hub are typically not executed directly from the ROM, but instead are shadowed or copied to and executed from a portion of the main memory. Thus, driver programs accessing the table need not necessarily directly access the ROM, but preferably access that table as it appears in the portion of main memory which reflects the ROM content. Moreover, a driver need not access the data table 44 to ascertain the services number every time, but only need access the data table 44 prior to the first call of the BIOS routine. Thereafter, the correct services number for the BIOS routine is known (if it was saved from a previous access).

[0033] Identifying BIOS routine service numbers in this way thus alleviates the requirement for each OEM to have one person responsible for assigning service numbers for the OEM value added BIOS routines. Further, programmers of kernel mode programs that require access to BIOS routines need not perform the prior art technique of simply calling the routine while being unsure that the particular routine is even implemented, or having the hard-coded decision tables which rely on the ROM code and ROM dates as a tangential indication of whether the particular ROM supports the BIOS routine desired. Rather, the driver programmer need only know the GUID of the particular BIOS routine desired, and ascertaining whether that BIOS routine is supported, and if so its service number, is a look-up operation in the table 44.

[0034] The above discussion is meant to be illustrative of the principles and various embodiments of the present invention. Numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. For example, the preferred embodiments describe the use of the GUIDGEN.EXE program to generate a number to uniquely identify each new BIOS routine. However, one of ordinary skill in the art, now understanding the concept of assigning a unique identifier to each BIOS routine, could design equivalent systems for generating unique identifiers, including systems that do not rely on the GUIDGEN.EXE program. Further, the preferred embodiments describe that the table that correlates the GUIDs to the service numbers resides on the BIOS ROM 42. However, this table need not necessarily be stored in the BIOS ROM, and any non-volatile storage location within the computer, for example in non-volatile RAM, may be equivalently used to store the table. Further, table 44 is described to correlate GUIDs to services numbers used as the Arg0 parameter when calling SAL_PROC in the preferred embodiments; however, in cases where the calling procedure is through traditional methods, the table 44 may also include the BIOS category in addition to the services number. Although in the preferred embodiments kernel mode programs such as drivers access the table that correlates the GUIDs to the service numbers, any software program that needs to execute a BIOS routine may utilize the table for determining service numbers prior to execution of the desired BIOS routine. Although it is envisioned that service numbers associated with particular BIOS routine GUIDs will be the same across multiple computers, this need not necessarily be the case. Because each driver program preferably determines a service number based on a GUID prior to calling the BIOS routine, the services numbers could vary from computer to computer. Finally, the preferred embodiments are described in a context of an Intel Itanium™ processor and chipset system; however, the structures and methods described herein are applicable in any computer system having any number of variety of microprocessor and related chipsets. It is intended that the following claims be interpreted to embrace all such variations and modifications.